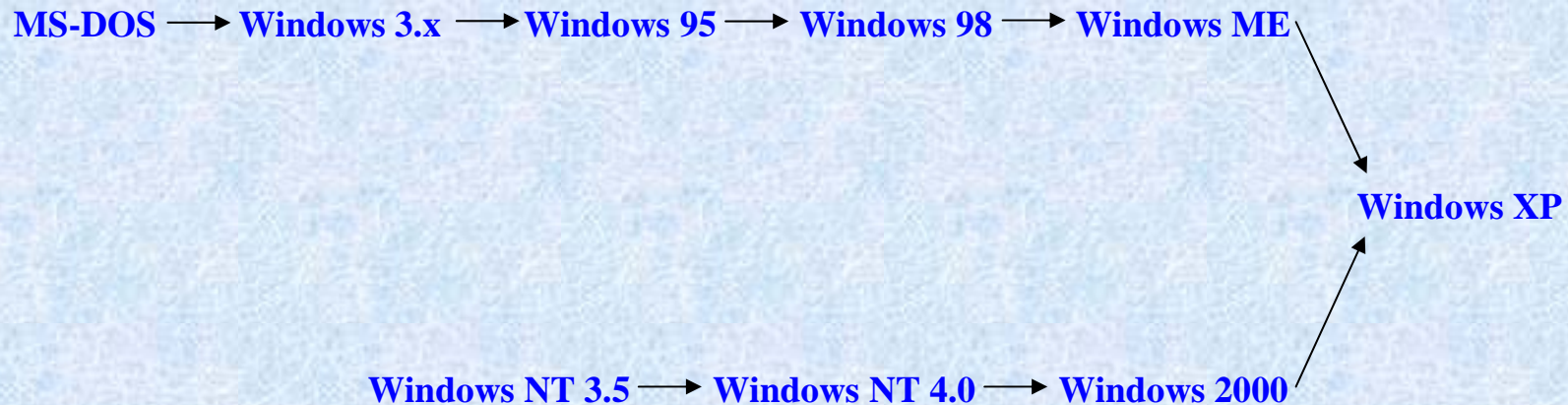


Il sistema operativo Windows

- 8.1 **Struttura generale**
- 8.2 **Gestione dei processi e dei thread**
- 8.3 **Sincronizzazione tra thread**

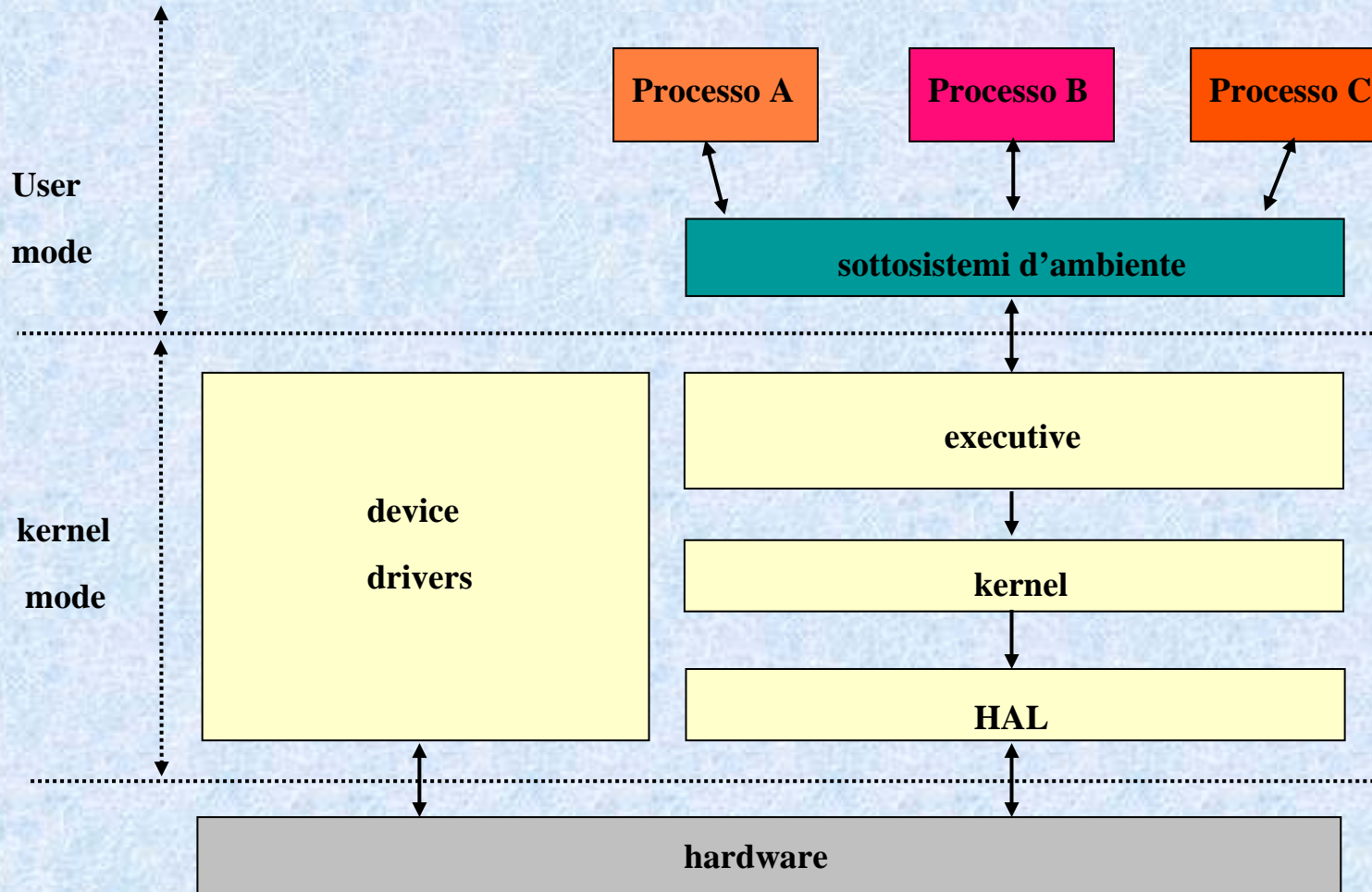
Storia di Windows

■ Evoluzione storica dei sistemi microsoft Windows



Struttura generale

■ Struttura modulare di Windows



Processi e thread

- Un processo in Windows è un “oggetto di nucleo” (*kernel object*) definibile come un "contenitore di risorse". È caratterizzato da:
 - ✓ Un identificatore
 - ✓ Uno spazio di indirizzamento privato
 - ✓ Una directory corrente
 - ✓ Varie tabelle contenenti le risorse del processo
 - ✓ Uno o più thread di esecuzione

Processi e thread

- Un thread è un “oggetto di nucleo” che definisce una entità concorrente e schedulabile.
È caratterizzato da:
 - ✓ Un identificatore
 - ✓ Una funzione da eseguire
 - ✓ Un contesto (insieme di registri del processore)
 - ✓ Un puntatore allo stack

Fibre (thread a livello utente)

- Oltre ai thread, che sono “oggetti del nucleo”, a livello utente si possono realizzare anche le **Fibre**.
- Le Fibre sono thread a livello utente, caratterizzati da:
 - ✓ Un identificatore
 - ✓ Una funzione da eseguire
 - ✓ Un contesto (insieme di registri del processore)
 - ✓ Uno stack
 - ✓ Un algoritmo di scheduling

Realizzazione mediante librerie a livello utente

Scheduling dei thread

- **Politica di scheduling dei thread: combina la politica a priorità con quella “round-robin”**
- **La priorità di un thread viene calcolata come somma di due componenti:**
 - ✓ **Una classe di priorità associata al processo a cui il thread appartiene**
 - ✓ **Una priorità relativa del thread**

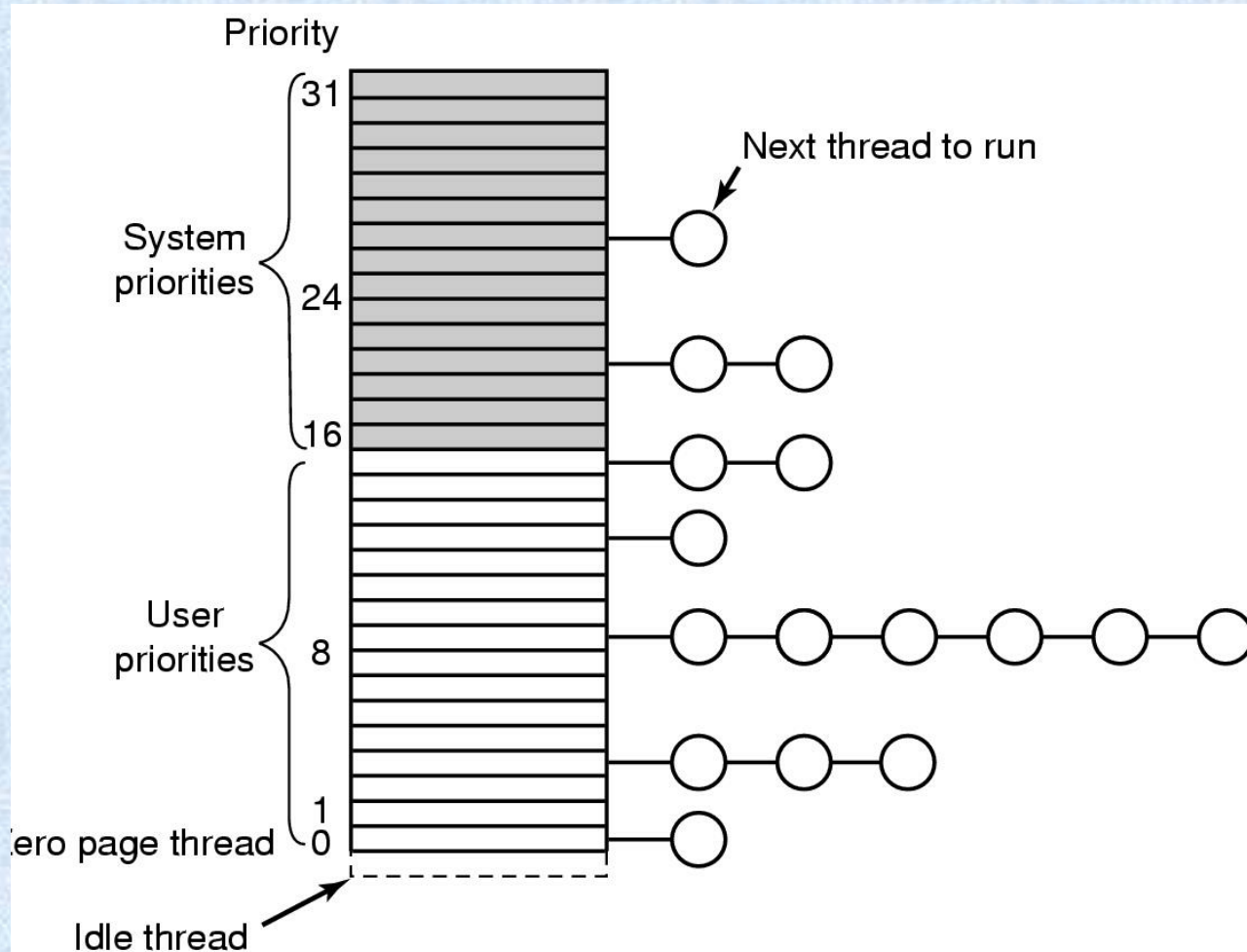
Scheduling dei thread

- **Classi di priorità di un processo (in ordine decrescente):**
 - ✓ **Real-time**
 - ✓ **High**
 - ✓ **Above Normal**
 - ✓ **Normal**
 - ✓ **Below Normal**
 - ✓ **Idle**

Scheduling dei thread

- **Priorità relative di un thread (in ordine decrescente):**
 - ✓ **Time Critical**
 - ✓ **Highest**
 - ✓ **Above Normal**
 - ✓ **Normal**
 - ✓ **Below Normal**
 - ✓ **Lowest**
 - ✓ **Idle**

Scheduling dei thread



Scheduling dei thread

Algoritmo di scheduling :

Si esegue il primo thread della prima coda non vuota per massimo 1 *quanto* (20ms--120ms)

Scheduling round robin fra thread con la stessa priorità

Come variano le priorità nel tempo :

- i thread tipicamente entrano a priorità 8
- la priorità viene elevata se:
 - viene completata una operazione di I/O (+1 disco, +2 linea seriale, +6 tastiera, +8 scheda audio ...)
 - termina l'attesa su un semaforo, mutex, evento (+1 background, +2 foreground)
 - l'input nella finestra di dialogo associata al thread è pronto

Scheduling dei thread

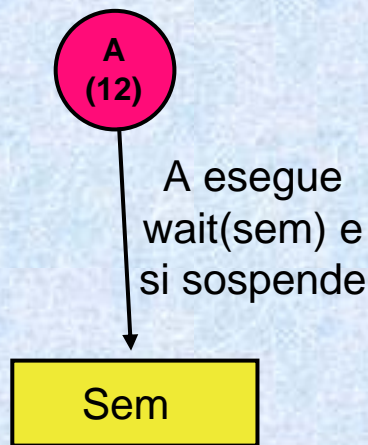
Algoritmo di scheduling :

Come variano le priorità nel tempo (cont.):

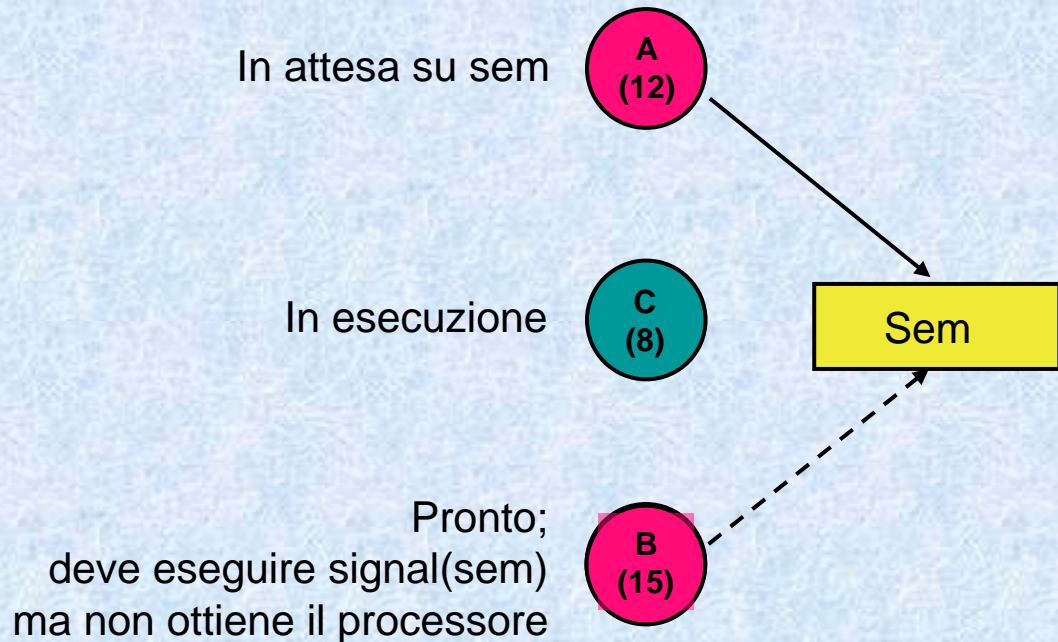
- la priorità viene abbassata se:
 - un thread usa tutto il suo quanto (-1)
- se un thread non ha girato per un tempo maggiore di una soglia fissata, allora passa per 2 quanti a priorità 15 (serve a gestire potenziali inversioni di priorità)

Quando una finestra va in foreground il quanto dei thread corrispondenti viene allungato

Scheduling dei thread



1)



2)

Un esempio di inversione di priorità

Meccanismi di sincronizzazione

- Esistono due tipi di meccanismi di sincronizzazione:
 - ✓ Un insieme di meccanismi “leggeri”: *funzioni interlocked* (es. spin-lock), riservati alle fibre
 - ✓ Un insieme di meccanismi per sincronizzazioni più complesse: tramite “oggetti di nucleo” (es. semafori, mutex, ecc.), riservati ai thread (dello stesso processo o di processi diversi)

Meccanismi di sincronizzazione

1) Riservati alle fibre e locali a un processo

- implementati con librerie nello spazio utente

Sezioni Critiche

- `EnterCriticalSection()` / `LeaveCriticalSection()`

Meccanismi di sincronizzazione

1) Riservati ai thread; locali a un processo

- implementati nel kernel

Mutex

- **ReleaseMutex()** corrisponde alla *signal*
- **WaitForSingleObject()** corrisponde alla *wait*

Eventi

- due stati *set / cleared*
- **WaitForSingleObject()** attesa su evento
- **SetEvent()** segnala che l'evento si è verificato

Semafori

- creati con **CreateSemaphore()**
- **ReleaseSemaphore()** corrisponde alla *signal*
- **WaitForSingleObject()** corrisponde alla *wait*

Meccanismi di sincronizzazione

- 3) Riservati ai thread; per la comunicazione tra thread di processi diversi
- **implementati nel kernel**

Pipe

Sono previsti diversi tipi :

- **byte** : funzionano come in Unix
- **message** : preservano i limiti dei singoli messaggi
- **named pipe** : possono essere utilizzate anche in rete

Mailslots

- simili ai pipe
- permettono di aver più destinatari e di inviare messaggi in broadcast